VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Detekce obsazenosti parkovacích míst pomocí algoritmu strojového učení s učitelem

Parking Lot Occupancy Detection by Supervised Machine Learning Algorithm

2014 Michal Šamaj

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Michal Šamaj**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Detekce obsazenosti parkovacích míst pomocí algoritmu strojového učení s učitelem
Parking Lot Occupancy Detection by Supervised Machine Learning Algorithm

Zásady pro vypracování:

V dnešní době je využíváno technik zpracování obrazu v mnoha oblastech dopravy. Jedním z problémů je i detekce obsazenosti parkovacích ploch. Získané informace je možno využít např. pro automatizované parkovací systémy nebo v inteligentních dopravních systémech. Cílem práce je vytvořit aplikaci pro detekci volných parkovacích míst. Detekci proveďte pomocí vybraného algoritmu strojového učení s učitelem. Ve své práci proveďte:

1. Popište zadaný problém.
2. Představte princip strojového učení s učitelem.
3. Analyzujte řešení a popište potřebnou teorii vybraného algoritmu.
4. Implementujte aplikaci s použitím knihovny OpenCV.
5. Otestuje výslednou aplikaci a zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

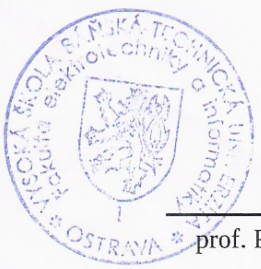
[1] Duda, Hart, Stork: Pattern Classification, 2000, ISBN: 978-0471056690
[2] Mohri, Rostamizadeh, Talwalkar: Foundations of Machine Learning, 2012, ISBN: 978-0262018258

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michael Holuša**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

V Ostrave 5. mája 2014

.....

Rád by som na tomto mieste poďakoval pánovi Ing. Michaelovi Holušovi, ktorý ma vďaka plnohodnotným konzultáciám viedol k úspešnému zvládnutiu tejto práce.

Abstrakt

V dnešnej dobe sa využívajú techniky spracovania obrazu v mnohých oblastiach dopravy. Jedným z problémov je i detekcia obsadenosti parkovacích plôch. Získané informácie je možné využiť napr. pre automatizované parkovacie systémy alebo v inteligentných dopravných systémoch. Práca popisuje problém detekcie voľných parkovacích miest a predstavuje princíp strojového učenia s učiteľom. V práci bol vybraný a popísaný jeden z algoritmov pre strojové učenie s učiteľom. Následne bol pomocou knižnice OpenCV implementovaný a v konečnej fázi aj otestovaný. V práci sú popísane dosiahnuté výsledky.

Kľúčové slová: spracovanie obrazu, detekcia obsadenosti parkoviska, algoritmus strojového učenia s učiteľom

Abstract

Nowadays, image processing techniques are used in many fields of transport. One of the problems is the occupancy detection of parking areas. The obtained information can be used for example, for automated parking systems or in intelligent transport systems. This thesis describes the problem of detection of free parking spaces and introduces the principle of supervised machine learning. One of the algorithms for supervised machine learning was selected and described. Subsequently it was implemented using OpenCV library and also tested at the final stage. The obtained results are described.

Keywords: image processing, parking occupancy detection, supervised machine learning algorithm

Zoznam použitých skratiek a symbolov

OpenCV	–	Open Source Computer Vision Library
HOG	–	Histogram of Oriented Gradients
SVM	–	Support Vector Machine
ROI	–	Region of Interest
RGB	–	Red Green Blue Color Model
2D	–	Two dimensional
3D	–	Three dimensional
BSD	–	Berkeley Software Distribution

Obsah

Zoznam tabuliek

Zoznam obrázkov

1 Úvod

Človek vníma svoje okolie pomocou zmyslov, pričom najväčší podiel na získavaniu podnetov z vonkajšieho prostredia má zrak. Ľudské oko vie podnet okamžite spracovať, vyhodnotiť ho a dokáže tak napríklad určiť, ktoré parkovacie miesto je obsadené a ktoré nie. Pre počítač je to však veľmi zložitá úloha, pretože nemá žiadne zmysly. Preto vzniklo odvetvie výpočetnej techniky, ktoré sa nazýva počítačové videnie. Zaoberá sa vytváraním aplikácií, ktoré sú schopné získať informácie zo zachyteného obrazu.

Dôležitou súčasťou počítačového videnia je detekcia objektov v obraze, s ktorou sa v dnešnej dobe stretávame pri detekcii ľudí, automobilov alebo dokonca budov. Vzhľadom na nedostatok parkovacích miest v mestách je v súčasnosti pre človeka dôležité, čo najrýchlejšie zistiť dostupnosť voľného parkovacieho miesta. Riešenie tohto problému ponúka práve detekcia dostupnosti parkovacích miest. Informácie získané prostredníctvom tejto detekcie je potom možno využiť v inteligentných parkovacích domoch alebo automatizovaných parkoviskách, ktoré nepotrebujú ľudskú obsluhu. Aby však rozpoznávanie dostupnosti parkovacieho miesta bolo časovo a technicky menej naročné, je treba zvoliť správny algoritmus. Preto je na mieste skúmať algoritmy strojového učenia, ktoré sa pri detekcii obsadenosti využívajú.

Cieľom práce je teoretický popis detekcie objektov v obraze a princíp strojového učenia s učiteľom. Dôležité je aj popísať vybratý algoritmus strojového učenia a naimplementovať výslednu aplikáciu s využitím knižnice OpenCV, následne ju otestovať a zhodnotiť dosiahnuté výsledky.

Prvá kapitola je úvodom do problematiky a druhá kapitola sa zaoberá popisom existujúcich riešení problému. V tretej kapitole je krátky popis knižnice OpenCV, ktorej funkcie sa v tejto práci používajú. Následujúce časti popisujú techniky predspracovania obrazu, ktoré je dôležitou súčasťou spracovania obrazu. Vo štvrtej kapitole je popis odstránenia skreslenia šošovky kamery a priebeh transformácie, ktorá zmenila perspektívu pohľadu na parkovisko, spoločne so získaním jednotlivých parkovacích miest. Piata kapitola sa zameriava na métody detekcie objektov v obraze a teoreticky popisuje techniku *histogramu orientovaných gradientov*, ktorý je použitý vo výslednej aplikácii. Šiesta kapitola je venovaná strojovému učeniu a popisu algoritmu *support vector machine*, ktorý je použitý ako klasifikátor. Kapitoly sedem a osem opisujú implementáciu, testovanie aplikácie a zhodnotenie dosiahnutých výsledkov.

2 Existujúce riešenia problému

V súčasnosti takmer žiadne parkovisko nemá systém kontroly obsadenosti. Väčšina z nich je obsluhovaná človekom, čo je niekedy nedostatočné. Problém, ktorý sa vždy vyskytuje, je že ľudia nechcú stráviť veľa času hľadaním voľného parkovacieho miesta. Nemajú jednoducho čas krúžiť dookola parkoviska, kým sa nejaké miesto uvoľní. Tento problém sa výrazne zhoršuje v husto zaľudnených oblastiach, kde sa vyskytuje veľa ľudí a automobilov.

Bolo už vynájdených niekoľko systémov, od jednoduchých manuálnych implementácií až po automatizované počítačové systémy. Súčastným najvyužívanejším riešením je použitie závory a parkovacích lístkov. V poslednom období sa skôr začína s využívaním počítačových systémov, a preto sa mnoho vedcov a inžinierov zaoberá vymýšľaním a implementáciou takéhoto systému. V posledných rokoch bolo vyvinutých niekoľko metód detekcie objektov v obrazu, ktoré využívajú algoritmus strojového učenia [3, 4, 5]. Systémy detekcie objektov môžeme vo všeobecnosti rozdeliť do niekoľko skupín: algoritmy pre detekciu príznakov, algoritmy pre detekciu pomocou modelov, algoritmy založené na odstránení pozadia.

Metódy implementované na detekciu automobilov môžeme rozdeliť do dvoch tried[6]: na explicitné alebo implicitné.

Explicitné metódy [7] používajú generické modely automobilov. Reprezentujú ich ako 2D alebo 3D model. Tieto modely sa častokrát ďalej rozdeľujú na čiastkové modely, a tie sú detekované nezávislé podľa ich vlastností. Ich kombinácou sa potom detekuje celý automobil.

Implicitné metódy [8] sú založené na vzhľade automobilov. Používajú sa tu vzorové obrázky, ktoré popisujú ako vyzerá model automobilu, a tiež klasifikátory, použité na klasifikáciu vstupných vzorových obrázkov na základe získaných príznakov. Pre detekciu sa používajú algoritmy detekcie príznakov a algoritmy strojového učenia s učiteľom.

Systémy pre detekciu obsadenosti potrebujú niekoľko zdrojov informácií. Dôležité je na jednej strane zaistiť, tak veľké množstvo dát, ako je to možné, ale na druhej strane taktiež zaistiť vysokú spoľahlivosť týchto dát. Na získavanie týchto dát sa využívajú rôzne senzory, ale je to nepraktické, pretože je potrebné zaistiť príslušný senzor pre každé parkovacie miesto. Kvôli tomu môže byť pri veľkých parkoviskách takéto riešenie finančne náročné. Výhodnejšie je použitie kamerového systému, ktorému častokrát stačí jedna kamera na pokrytie celého parkoviska. Pri jeho použití je najčastejším riešením detekcia

automobilov v obraze. Taktiež sa využívajú metódy, kde sa detekuje stav parkovacieho miesta a používajú sa tu algoritmy na odstránenie pozadia.

Použitie kamerového systému prináša tiež iné výhody, ako napríklad zvýšenie účinnosti celého systému. Senzory umiestnené v asfalte parkoviska nemusia detekovať napríklad zaparkovaný motocykel. Alebo iným problémom môžu byť vodiči, ktorí nerešpektujú pravidlá, a zaparkujú svoj automobil naprieč cez čiary parkoviska. Ani v tomto prípade si senzory neporadia.

Obr. 1: Príklad použitia senzorového systému[19]

Obr. 2: Príklad použitia kamerového systému[18]

3 Knižnica OpenCV

Knižnica OpenCV obsahuje programátorské funkcie, ktoré sú predovšetkým určené na real-timeové spracovanie obrazu. Úplný význam skratky OpenCV je Open Source Computer Vision Library. To znamená, že táto knižnica je otvorená a voľne dostupná pre používateľov pod licenciou BSD. Pôvodne bola vyvinutá firmou Intel, známym výrobcom počítačového hardwaru, ale v dnešnej dobe sa na jej vývoji podieľajú firmy Willow Garage a Itseez. Firma Willow Garage sa zaoberá výrobou robotov a ich softwarového vybavenia. OpenCV je multiplatformná knižnica. Je napísaná v programovacom jazyku C++, ktorý je primárne určený pre toto prostredie. Existujú však aj rozhrania pre programovacie jazyky C, Python, Java a MATLAB/OCTAVE a pre všetky spomínané jazyky existuje aj online dokumentácia. OpenCV podporuje operačné systémy Windows, Linux, Android a MacOS.

Hlavné ciele projektu sú:

- **zaistenie pokročilého výskumu v oblasti počítačového videnia** tým, že poskytne nielen otvorený, ale predovšetkým optimalizovaný kód pre základnu infraštruktúru počítačového videnia,
- **ľahšie šírenie vedomostí o počítačovom videní** tým, že poskytne spoločnú infraštruktúru, na ktorej môžu vyvojári stavať, pretože kód je ľahšie čitateľný a prenositeľný
- **výroba pokročilých komerčných aplikácií počítačového videnia** a zaistenie ich prenositeľnosti a výkonnostne optimalizovaného kódu, ktorý je dostupný zadarmo, s licenciou, ktorá je voľne dostupná pre programátorov

V aplikácii, ktorá je súčasťou tejto práce sa knižnica OpenCV využíva na načítavanie a ukladanie obrázkov, ďalej na vykonanie perspektívnej transformácie a na získavanie jednotlivých parkovacích miest. Hlavné využitie je však pri implementácií HoG deskriptora, ktorý slúži na detekciu objektov v obraze a tiež pri implementácii SVM, ktorý slúži na klasifikovanie obsadenosti parkovacích miest.

4 Techniky predspracovania obrazu

Cieľom predspracovania obrazu je odstránenie šumu, skreslenia obrazu a zvýraznenie ostatných vlastností, ktoré sú potom pri samotnom spracovaní obrazu kľúčové. V tejto práci bolo dôležité odstrániť skreslenie šošovky kamery, aby sa následne mohla vykonať transformácia, pri ktorej došlo k zmene perspektívy pohľadu na parkovisko. Výsledkom bol obraz, na ktorom sú všetky parkovacie miesta rovnakej veľkosti. Sú to oblasti, z ktorých sa potom získavali informácie pre výslednú aplikáciu, ale tento problém je popísaný v ďalších kapitolách. V nasledujúcich podkapitolách sú opísané jednotlivé kroky, ktoré boli vykonané. Obrázky boli získané z kamery, ktorá sa nachádza v areáli VŠB v Ostrave. Príklad získaného snímku je na obrázku 5.

4.1 Odstránenie skreslenia šošovky

Kamery existujú už celkom dlhú dobu. Napriek tomu, s príchodom lacnejších a ľahšie dostupných kamier sa vyskytli určite problémy. Kamery sa síce stali súčasťou každej domácnosti, ale za cenu významného skreslenia. Našťastie toto skreslenie je konštantné a pomocou kalibrácie a premapovania ho môžeme odstrániť.

Kamera použitá pre získavanie obrázkov, ktoré sa využivajú v tejto práci má tzv. *fisheye* šošovku. Tento typ šošovky zaznamenáva hemosférické obrazy, čo znamená, že sa snaží zachytiť nekonečne široký obraz do konečne širokého obrázku. Takéto skreslenie sa v angličtine označuje ako *barrel distortion* [9], čo sa dá voľne preložiť ako skreslenie v tvare sudu a jeho príklad je na obrázku 3.

Skreslenie sa zvyšuje so vzdialenosťou od optických osí a najhoršie je v rohoch obrázku. Významnú úlohu zohráva aj zoom obrázku. Matematicky sa problém vyjadruje pomocou **Brownovho modelu skreslenia** [11]. Model bol naprogramovaný a pomocou *bilineárnej interpolácie* bolo skreslenie šošovky odstránené. Matematicky sa model vyjadruje nasledovne:

$$x_d = x_u(1 + K_1 r^2 + K_2 r^4 + ...) \tag{1}$$

$$y_d = y_u(1 + K_1 r^2 + K_2 r^4 + ...) \tag{2}$$

kde:

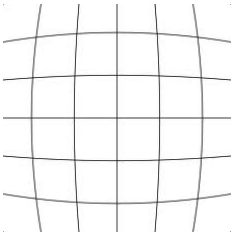
(x_d, y_d) = skreslené body obrázku, ktoré sú zadané podľa špefickej šošovky,

(x_u, y_u) = neskreslené body obrázku, ktoré sú zadané podľa ideálnej šošovky,

K_n = koeficient radiálneho skreslenia,

$r = \sqrt{x_u^2 + y_u^2}$ a

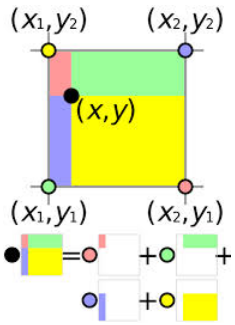
... = nekonečné opakovanie.



Obr. 3: Barrel distortion[20]

4.2 Bilineárna interpolácia

V počítačovom videní je bilineárna interpolácia[10] základnou prevzorkovaciou technikou. Keď sa na obrázok použije nejaký druh transformácie, je jasné, že dôjde k presunutiu niektorých pixelov na základe konštanty. Tým pádom ale na obrázku vzniknú prázdne pixely, ktoré nemajú priradenú žiadnu hodnotu z RGB spektra. Bilineárna interpolácia využíva na vyrátanie farebnej intenzity pixela iba 4 najblizšie hodnoty pixelov, ktoré sú umiestnené diagonálne od daného pixelu. Pre lepšiu demonštráciu si uvedieme obrázok:



Obr. 4: Bilineárna interpolácia[21]

V tejto geometrickej vizualizácii môžeme vidieť, že hodnota čierneho bodu je súčtom hodnoty každého farebného bodu, vynásobenou obdĺžnikovou oblasťou rovnakej farby.

Táto suma je vydelená celkovým obsahom všetkých 4 obdĺžnikov. Výsledok po odstránení skreslenia šošovky je zobrazený na obrázku 6.

Obr. 5: Pôvodný obrázok parkoviska

Obr. 6: Obrázok parkoviska po odstránení skreslenia

4.3 Perspektívna transformácia

Aby bolo možné z obrázku získať informácie o jednotlivých parkovacích miestach, a tým určiť či je miesto obsadené alebo volné, je potrebné najskôr vykonať na obrázku transformáciu, po ktorej budú mať všetky parkovacie miesta približne rovnakú veľkosť. Táto transformácia bola vykonaná pomocou funkcií z knižnice OpenCV.

Najskôr sa pomocou funkcie `getPerspectiveTransform` [12] získa perspektívna matica o rozmeroch 3x3. Táto matica sa získava pomocou 4 párov korešpondujúcich bodov, pričom jedna skupina sú koordináty z pôvodného obrázku a druhá skupina sú korešpondujúce body z cieľového obrázku. Matematicky môžeme perspektívnu maticu vyjadriť nasledovne:

$$dst[t_i x_i, t_i y_i, t_i] = M * src[x_i, y_i, 1], \tag{3}$$

kde body na cieľovom obrázku dostaneme prenásobením bodov na pôvodnom obrázku, pričom i je v intervale od 0 po 3.

Potom sa pomocou funkcie `warpPerspective` [12] transformuje pôvodný obrázok pomocou vyššie spomínanej matice. Matematické vyjadrenie transformácie je nasledovné:

$$dst(x, y) = src(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}) \tag{4}$$

Rovnica vyjadruje, ako sa vyráta hodnota konkrétneho pixelu. Indexy ukazujú, na ktorom mieste v perspektívnej matici je umiestenená použitá hodnota. Obrázok 7 zobrazuje parkovisko po aplikovaní perspektívnej transformácie.

Obr. 7: Obrázok parkoviska po perspektívnej transformácii

4.4 Získanie jednotlivých parkovacích miest

Keď už sa na obrázok aplikuje aj perspektívna transformácia, je potrebné pristúpiť k poslednému kroku predspracovania obrazu, a to k získaniu jednotlivých parkovacích miest. Využije sa na to funkcia `setImageROI`[12] z knižnice OpenCV. Po zadaní počiatočných bodov parkovacieho miesta, ktoré sa získajú z obrázku a zadaní jeho výšky a šírky, dostaneme ako návratovú hodnotu výrez obrázka. Z takého obrázka sa potom získavajú potrebné informácie, ktoré sa používajú vo výslednej aplikácii a ich získavaniu a následnému spracovaniu sú venované následujúce kapitoly. Takýto prístup k získaniu parkovacieho miesta si vyžaduje ľudský zásah a namapovanie presných hodnôt.

5 Detekcia objektov v obraze

V počítačovom videní a spracovaní obrazu koncept tzv. *príznakovej detekcie* (z ang. *feature detection*) [1]odkazuje na metódy, ktoré sa zaoberajú výpočtom abstrakcie obrazových informácií a vytvárajú v každom bode obrazu rozhodnutie, či sa tam určitý obrazový príznak nachádza alebo nie. Výsledné príznaky sú podmnožinami obrazu, často vo forme izolovaných bodov, kontinuálnej krivky alebo prepojených oblastí.

Neexistuje presná definícia ako v našom jazyku vyjadriť výraz *feature* a definícia často závisí od konkrétneho použitia alebo aplikácie. V tejto práci bude definovaný názov príznak, pretože sa bude vyhľadávať určitá zaujímavá vlastnosť obrazu, ktorá bude potom použitá ako východiskový bod pre algoritmus strojového učenia. Vzhľadom k tomu, že vektory týchto zaujímavých príznakov sú použité ako východiskové body sú tieto algoritmy tak dobré, ako je dobrá detekcia týchto príznakov. V dôsledku toho je dôležitou vlastnosťou detekcie príznakov opakovanie: či je alebo nie je rovnaký príznak detekovaný v dvoch alebo viacerých rôznych snímkoch rovnakej scény.

Detekcia príznakov obrazu je nízko-urovňová operácia spracovania obrazu. To znamená, že sa zvyčajne vykoná ako prvá operácia pri spracovaní obrazu a skúma každý pixel obrázku, aby zistila, či sa daný príznak v pixeli nachádza alebo nie. Ak je súčastou väčšieho algoritmu, tak typicky skúma len určité regióny obrázku a vyhľadáva daný príznak. Pre dosiahnutie lepších výsledkov sa môže obrázok vyhladiť pomocou Gaussových funkcií v mierke priestorovej reprezentácie. Samotný príznakov alebo niekoľko príznakov, najčastejšie pomocou lokálnych derivácií.

Množstvo algoritmov počítačového videnia používa detekciu príznakov ako základnú vec, a preto vzniklo množstvo algoritmov pre detekciu príznakov. Tie sa veľmi líšia vo vlastnostiach, ktoré získavajú, vo výpočtovej zložitosti a opakovateľnosti. Preto ich môžeme rozdeliť do následujúcich skupín:

- **Hrany** - sú to miesta, kde je hranica medzi dvoma regiónmi na obraze. Všeobecne je známe, že okraj môže mať ľubovoľný tvar a môže obsahovať spojenie. V praxi sú hrany definované ako miesta v obraze, ktoré majú silný gradient.

- **Rohy** - používajú sa tu algoritmy, ktoré najskôr detekujú hrany a potom ich analyzujú aby detekovali ich rapídne zmeny, tzv. rohy. V praxi sa opäť označujú ako zakrivenia s veľkou hodnotou gradientu.

- **Regióny/body záujmu** - tieto detektory sa používajú na detekovanie oblastí, ktoré sú pre vyššie spomínané dva algoritmy príliš hladké.
- **Hrebene** - možno ich hápať ako zovšeobecnenie mediálnej osi. Z praktického hľadiska si ho môžeme predstaviť ako jednorozmerné krivky, ktoré predstavujú osy symetrie, a nesú aj informáciu o šírke hrebeňa v aktuálnom mieste.

5.1 Histogram orientovaných gradientov

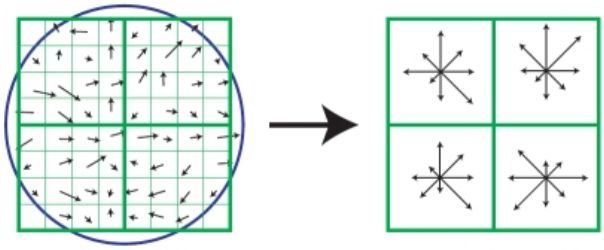
Histogram orientovaných gradientov je deskriptor príznakov používaný v počítačovom videní a spracovaní obrazu pre účely detekcie objektov.

Základná myšlienka histogramu orientovaných gradientov je, že lokálny vzhľad a tvar objektu v obraze možno popísať rozložením intenzity gradientov alebo detekciou hrán. Implementácia tohto deskriptoru sa zvyčajne uskutočňuje pomocou rozdelenia obrazu do malých prepojených oblastí, ktoré sa nazývajú **bunky**, a v každej bunke sa zostaví histogram orientovaných gradientov alebo orientovaných hrán pre každý pixel. Kombinácia týchto histogramov potom reprezentuje deskriptor. Pre lepšiu presnosť, lokálne histogramy môžu byť normalizované pomocou kontrastu, a to tak, že sa vyráta jeho miera intenzity vo väčšej oblasti obrázku, ktorý sa nazýva **blok** a potom sa pomocou tejto hodnoty štandardizujú všetky hodnoty v bloku. To má za následok lepšie invarianty zmien v osvetlení alebo zatienení.

HoG deskriptor má niekoľko kľúčových výhod oproti iným metodám deskriptorov. Pretože HoG deskriptor pracuje s lokalizovanými bunkami, podporuje tak invarianty geometrických a fotometrických transformácií, s výnimkou orientácie objektov. Tieto zmeny sa prejavia až vo väčších priestorových regiónoch. HoG deskriptor sa predovšetkým používa pri detekcií ľudí v obrazoch, ale môže sa použiť aj v iných oblastiach [13].

5.1.1 Implementácia algoritmu

1. **Výpočet gradientu** - Prvým krokom výpočtu v predspracovaní obrazu pre detektory príznakov je zabezpečenie normalizovaných farieb a gamma hodnôt. Tento krok však môže byť pri HoG deskriptore vynechaný, pretože samostná normalizácia, ktorú tento deskriptor sám o sebe prevedie dosahuje rovnaký výsledok. Preto aktuálny stav obrázku neposkytuje výrazný vplyv na rýchlosť výpočtu.
   Namiesto toho sa ako prvý krok prevedie výpočet hodnôt gradientu. Najbežnejšou metódou je jednoducho aplikovať jednorozmerné centrované pole diskrétnych bodov odvodených z masky v jednom alebo obidvoch horizontálnych a vertikálnych smeroch. Konkrétne táto metóda vyžaduje filtrovanie farieb alebo intenzity dát z obrázku pomocou nasledujúcich filtrovacích kernelov:
   $$[-1, 0, 1], [-1, 0, 1]^T. \tag{5}$$

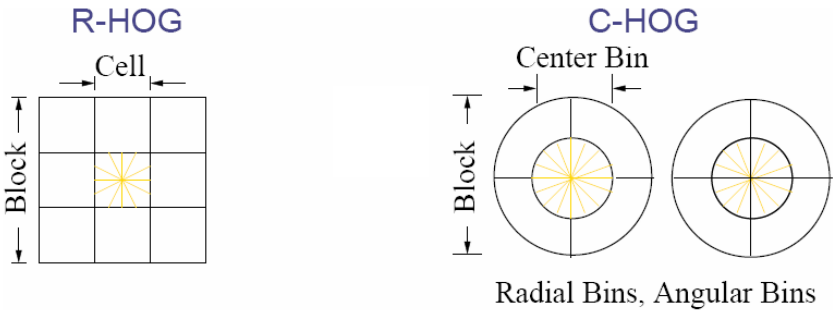


Obr. 8: Deskriptory histogramu orientovaných gradientov[22]

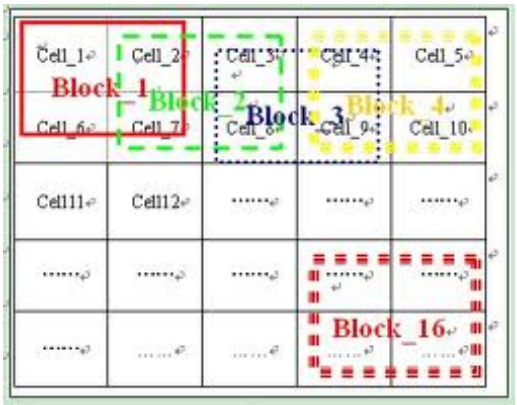
2. **Orientácia obrazových bodov** - Druhý krok výpočtu zahŕňa vytvorenie bunkového histogramu. Každý pixel v bunke hrá významnú rolu pre orientačne založené histogramové kanály, ktoré sú založené na hodnotách nájdených pri výpočte gradientu. Samotné bunky môžu mať obdĺžnikový alebo radiálny tvar a samostatné kanály histogramu môžu byť rovnomerne rozdelené od 0 po 180 stupňov alebo od 0 po 360 stupňov. Záleží to na tom či je gradient znamienkový alebo bezznamienkový. V praxi sa zistilo, že bezznamienkové gradienty použité spoločne s deviatimi histogramovými kanálmi dosahujú najlepšie výsledky pri detekcii objektov v obraze.

3. **Bloky deskriptora** - Aby sa zohľadnili zmeny v osvetlení a kontraste, sila gradientu musí byť lokálne normalizovaná, čo si vyžaduje zoskupenie buniek dokopy

do väčších, priestorovo prepojených blokov. HoG deskriptor je potom vektor zložiek normalizovaných buniek histogramu zo všetkých regiónov bloku. Tieto bloky sa zvyčajne prekrývajú, čo znamená, že každá bunka sa v konečnom deskriptore prejaví viacej ako raz. Existujú dva hlavné geometrické tvary bloku: obdĺžinkové $R - HoG\ bloky$ a kruhové $C - HoG\ bloky$. R-HoG bloky sú zvyčajne štvorcové mriežky, reprezentované 3 parametrami: *počet buniek na blok*, *počet pixelov na bunku*, *počet kanálov na bunkový histogram*. C-HoG bloky môžeme nájsť v 2 variantoch: tie, ktoré majú jednu centrálnu bunku a tie, kde sa centrálna bunka uhlovo rozdeľuje. Okrem toho, tieto C-HoG bloky môžu byť popísané pomocou 4 parametrov: *počet uhlových a kruhových výrezov*, *polomer stredového výrezu*, *faktor zväčšenia polomeru pre ďalšie kruhové výrezy*.



Obr. 9: Bloky deskriptora[23]



Obr. 10: Rozloženie buniek a blokov[23]

4. **Normalizácia bloku** - Existujú 4 rôzne metódy normalizácie bloku. Nech v je nenormalizovaný vektor obsahujúci všetky histogramy daného bloku, $|v|_k$ bude jeho k-norma pre $k = 1, 2$ a e bude nejaká malá konštanta (jej presná hodnota nie je dôležitá). Potom normalizačný faktor môže byť jeden z nasledujúcich:

$$L2 - norm : f = \frac{v}{\sqrt{|v|_2^2 + e^2}} \tag{6}$$

$$L2 - hys : L2 - norma, kde\ v \leq 0.2 \tag{7}$$

$$L1 - norm : f = \frac{v}{\sqrt{|v|_1 + e}} \tag{8}$$

$$L1 - sqrt : f = \sqrt{\frac{v}{|v|_1 + e}} \tag{9}$$

   V praxi sa zistilo, že L2-hys, L2-norm a L1-sqrt metódy poskytujú podobný výkon, zatiaľ, čo L1-norma poskytuje o niečo menej spoľahlivý výkon. Avšak všetky štyri metódy poskytujú veľmi výrazné zlepšenie oproti nenormalizovaným dátam.

5. **SVM klasifikátor** - Posledným krokom v rozpoznávaní objektov pomocou deskriptora histogramu orientovaných gradientov je "nakŕmenie" nejakého rozpoznávacieho systému, dátami z deskriptora. Rozpoznávací systém môže byť založený na princípe strojového učenia s učiteľom. Support Vector Machine je binárny klasifikátor, ktorý hľadá optimálnu nadrovinu ako hlavný rozhodovací nástroj. Akonáhle sa naučí na obrázkoch, ktoré obsahujú určité objekty, môže SVM klasifikátor, prijímať rozhodnutia ohľadom prítomnosti objektu, ako sú napr. ľudské bytosti na testovacích obrázkoch.

5.2 HoG v OpenCV

V tejto práci bola použitá funkcia `HOGDescriptor` z knižnice OpenCV. Funkcia prijíma niekoľko parametrov, ktorými sa nastavujú vlastnosti deskriptora. Parametre sú: **win-size** - veľkosť detekčného okna, **block-size** - veľkosť bloku v pixeloch, **block-stride** - krokovanie bloku, **cell-size** - veľkosť bunky, **nbins** - počet kanálov, **win-sigma** - parameter pre nastavenie Gaussovho vyhladzovania, **threshold-L2hys** - maximálny parameter pre nastavenie L2hys normalizácie, **gamma-correction** - parameter na určenie či je potrebná gamma korekcia, **nlevels** - maximálny počet zväčsovania detekčného okna.

6 Strojové učenie

Strojové učenie [2] je odbor umelej inteligencie, ktorý sa týka výstavby a štúdie systémov, ktoré sa môžu učiť z dát. Napríklad, systém strojového učenia by sa mohol natrénovať na dátach z emailových správ, a tak sa naučiť rozlišovať medzi správami, ktoré obsahujú spam, a tými, ktoré nie. Po trénovacej fázi nastáva klasifikovanie, v ktorej sa môžu testovať emailové správy a klasifikátor ich rozdelí do zložiek.

Jadro strojového učenia sa zaoberá reprezentáciou a generalizáciou. Reprezentácia dát a funkcií, ktoré sa vypočítavajú nad týmito dátami, sú súčasťou každého systému strojového učenia. Generalizácia je vlastnosť systému, ktorá zaručuje, že systém bude fungovať aj na neviditeľných dátach. Podmienky, za ktorých sú splnené obe tieto vlastnosti, sú kľúčovým predmetom v oblasti teórie strojového učenia.

Existuje široká škála úloh strojového učenia a taktiež aj úspešných aplikácií. Optické rozpoznávanie znakov, v ktorom sú vytlačené symboly rozpoznávané podľa predchadzajúcich príkladov, je klasická reprezentácia strojového učenia.

Algoritmy strojového učenia môžu byť rozdelené na základe očakávaného výsledku alebo typu vstupných údajov v trénovacej fázi na:

- **Algoritmy strojového učenia s učiteľom** - trénovacie dáta sú označené, to znamená, že pri vstupe vieme, aký výstup očakávame. Pod dohľadom učiteľa sa algoritmus snaží zovšeobecniť funkciu alebo mapovanie vstupu na výstupy, ktoré môžu byť využité na generovanie výstupu z predtým nepoužitých vstupov.

- **Algoritmy strojového učenia bez učiteľa** - pracujú s neoznačenými dátami, to znamená, že pri vstupe je očakávaný výstup neznámy. Cieľom je zistiť štruktúru dát, napr. zhlukovou analýzou, neprevádza sa generelizácia dát a mapovanie vstupov na výstupy.

- **Algoritmy strojového učenia s poloičným učiteľom** - kombinujú oba princípy, pracujú s označenými aj neoznačenými dátami a generuje príslušnú funkciu alebo klasifikátor.

- **Transduktívne algortimy** - snažia sa predpovedať nové výstupy na základe konkrétnych a pevných prípadov zo sledovaných a špecifických vstupov.

6.1 Strojové učenie s učiteľom

Strojové učenie s učiteľom je úloha strojového učenia, ktorá odvodzuje funkcie z označených trénovacích dát. Trénovacie dáta sa skladajú zo súboru trénovacích príkladov. Každý príklad je tvorený párom, ktorý pozostáva zo vstupného objektu (najčastejšie vektor) a požadovanej výstupnej hodnoty. Algoritmus strojového učenia s učiteľom analyzuje trénovacie dáta a vytvára funkciu, ktorá sa potom používa na mapovanie nových príkladov. Optimálny algoritmus umožňuje priradiť správne výstupné hodnoty pre príklady, ktoré nikdy predtým nevidel. Od algortimu sa tiež vyžaduje, aby túto operáciu vykonával všeobecne a "rozumne".

Predtým než sa rozhodneme použiť algortimus strojového učenia s učiteľom, je potrebné uskutočniť nasledujúce kroky:

1. Presne určiť typ trénovacích príkladov. Napríklad, pre prípad analýzy rukopisov, by ako trénovacie dáta mohli poslúžiť jeden rukou napísaný znak, slovo alebo celý riadok.

2. Získať trénovací súbor. Tento súbor musí reprezentovať prvky reálneho sveta. Najčastejšie sa používajú výsledky experimentov alebo rôznych meraní.

3. Určiť akou formou budú reprezentované vstupné dáta. Typicky sa vstupný objekt transformuje na vektor, ktorý obsahuje veľký počet vlastností, ktoré presne popisujú objekt. Vektor, však nesmie obsahovať veľký počet dát, kvôli problému, ktorý je známy ako *preklatie viacrozmernosti*.

4. Zvoliť si typ algoritmu strojového učenia. Môžeme si vybrať napríklad *support vector machine* alebo *rozhodovacie stromy*.

5. Spustiť algoritmus strojového učenia na trénovacej množine. Niektoré algoritmy si vyžadujú nastavenie dodatočných parametrov. Sú to najčastejšie parametre validácie alebo optimalizácie.

6. Testovať presnosť naučeného algoritmu. Po nastavení parametrov by mala byť výstupná funkcia otestovaná na množine príkladov, ktoré nie sú zhodné s trénovacou množinou.

6.1.1 Implementácia algoritmu strojového učenia s učiteľom

Je daná trénovacia množina $\mathbb{N}$ vo forme $\{(x_1, y_1),\ \ldots, (x_N, y_N)\}$ tak, že x_i je vektor vlastností i-tého príkladu a y_i je jeho značka (trieda). Algoritmus potom hľadá funkciu $g : X \to Y$, kde X je vstupný priestor a Y je výstupný priestor. Funkcia g je súčasťou nejakého priestoru všetkých možných funkcií G, obvykle sa nazýva hypotézny priestor. Niekedy je výhodné funkciu g reprezentovať ako vyhodnocovaciu funkciu $f : X \times Y \to \mathbb{R}$ tak, že funkcia g je definovaná, aby vracala y s najväčšou hodnotou: $g(x) = \arg\max_y\ f(x, y)$. F je teda priestor vyhodnocovacích funkcií.

Aj keď G a F môžu byť hocijaké priestory funkcií, väčšina algoritmov strojového učenia sú pravdepodobnostné modely, kde g naberá hodnotu podmienenej pravdepodobnosti: $g(x) = P(x|y)$. Existujú dve metódy ako zvoliť funkcie g a f: *empirická minimalizácia rizika* a *štrukturálna minimalizácia rizika*. Empirická metóda hľadá funkciu, ktorá najlepšie vyhovuje vzhľadom na trénovacie dáta. Štrukturálna metóda obsahuje penalizácie, ktorými riadi rozptylový kompromis.

V oboch prípadoch sa predpokladá, že trénovacia množina pozostáva z nezávislých a rovnako rozdelených párov $(x_i,\ y_i)$. Aby bolo možné zmerať ako dobre sedí funkcia k trénovacím dátam je potrebné zadefinovať stratovú funkciu $L : Y \times Y \to \mathbb{R}^{\geq 0}$. Napríklad pre trénovací príklad $(x_i,\ y_i)$ je hodnota predpokladanej straty $\hat{y}$ vyjadrená pomocou $L(y_i,\ \hat{y})$.

Riziko $R(g)$ z funkcie g je definované ako predpokladaná strata g. Hodnotu možno odhadnúť z údajov z trénovacej fázy ako:

$$R_{emp}(g) = \frac{1}{N} \sum_i L(y_i,\ g(x_i)). \tag{10}$$

V **empirickej minimalizácii rizika** sa algoritmus strojového učenia snaží nájsť funkciu g, ktorá minimalizuje $R(g)$. Preto môže byť algoritmus strojového učenia implementovaný, tak aby používal optimalizačný algoritmus k nájdeniu g. Stratová funkcia $L(y,\ \hat{y})$ sa potom rovná $-\log P(y|x)$, čo je vlastne *metóda maximálnej vierohodnosti*.

Štrukturálna minimalizácia rizika slúži na odstránenie problému preučenia, ktorý sa môže vyskytnúť. Zavádza regulačnú penalizáciu, čím zaručuje optimalizáciu algoritmu. Populárnou formou penalizácie je vzťah $\sum_j \beta_j^2$, čo je vlastne druhá mocnina *Euklidovskej normalizácie váh*. Penalizácia sa označuje ako $C(g)$. Potom celková funkcia minimalizácie $J(g)$ má tvar $R_{emp}(g) + \delta C(g)$.

6.2 Support Vector Machine

V strojovom učení, sú *support vector machine* [14] samoučiace sa modely s učiteľom, ktoré spoločne s pridruženými algoritmami analyzujú dáta a rozpoznávajú vzory. Tie sa potom používajú pre klasifikáciu a regresívnu analýzu. Na začiatku je daná trénovacia množina príkladov, pričom každý z nich je označkovaný podľa príslušnosti do dvoch kategórií. SVM trénovací algoritmus vytvorí model, ktorý priraďuje nové príklady do jednej alebo druhej kategórie, a tým vytvára *nepravdepodobnostný binárny lineárny klasifikátor*. Model SVM je reprezentácia príkladov ako bodov v priestore, ktoré sú namapované tak, že príklady jednotlivých kategórií sú rozdelené do jasného rozdielu, tak aby hranica medzi nimi bola tak široká, ako je to len možné. Nové príklady sú potom namapované do rovnakého priestoru a predpokladá sa, že patria do katégorie podľa toho, na ktorú stranu medzery spadajú.

Okrem lineárnej klasifikácie, SVM môže efektívne vykonávať aj nelineárne rozhodovanie za použitia *kernel trick*, čím sa dáta implicitne mapujú do viac dimenzionálnych priestorov.

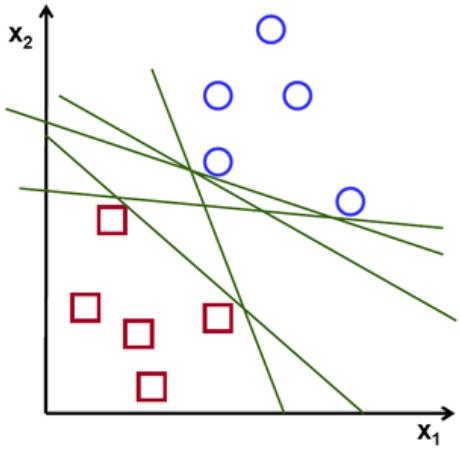
Support Vector Machine vytvára *nadrovinu* alebo niekoľko nadrovín vo viac alebo nekonečne dimenzionálnom priestore, ktoré môžu byť použité pre klasifikáciu, regresiu alebo ostatné úlohy. Dobrá seperácia je dosiahnutá takou nadrovinou, ktorá má najväčšiu vzdialenosť od najbližšieho bodu trénovacích dát hociktorej kategórie(tzv. funkčná hranica z ang. *function margin*), preto všeobecne znamená väčšia hranica menšiu chybovosť pri generelizácii klasifikátora.

Aj napriek tomu, že pôvodný problém môže byť vyjadrený pomocou konečného dimenzionálneho priestoru, často sa stáva, že dáta z tohto problému sa nedajú v tomto priestore lineárne oddeliť. Z tohto dôvodu bolo navrhnuté, že pôvodný konečne rozmerný priestor sa namapuje do viac dimenzionálneho priestoru, čo zlepšuje pravdepodobnosť rozdelenia v tomto priestore. Aby sa zachovala primeraná výpočetná záťaž, tak mapovanie použité v SVM je navrhnuté tak, aby bolo zabezpečené, že výpočet skalárneho súčinu je jednoduchší pre premenné z pôvodného priestoru tým, že sa definuje správna *kernel funkcia* $K(x, y)$ pre daný problém[15]. Nadroviny vo viac dimenzionálnom priestore sú definované ako súbor bodov, ktorých *skalárny súčin* s vektorom v tomto priestore je konštantný. Vektory definujúce nadroviny môžu byť vybraté tak, aby to boli lineárne kombinácie s parametrom α_i vektorov vlastností obrázkov z databázy. Pri tejto voľbe nadroviny, sa body x v priestore, mapujú do nadroviny podľa vzťahu $\sum_i \alpha_i * K(x_i, x)$

= *const*. Za povšimutie stojí, že ak $K(x, y)$ naberá malé hodnoty práve vtedy, keď sa y vzdiaľuje od x, tak každá zložka súčtu vyjadruje stupeň blízkosti testovacieho bodu x a korešpondujúceho bodu x_i z trénovacej fázy. Týmto spôsobom, súčet vyššie spomínaných kernelov môže byť použitý na meranie relatívnej vzdialenosti každého testovacieho bodu a trénovacieho bodu patriaceho do jednej alebo druhej skupiny, ktorá sa má rozlíšiť. Za povšimnutie tiež stojí, že množina bodov x mapovaných do každej nadroviny, môže mať celkom spletitý výsledok, čo umožňuje komplexnejšie rozlíšenie medzi súbormi, ktoré nie sú tak konvexné v pôvodnom priestore.

Klasifikácia dát je hlavnou náplňou strojového učenia. Napríklad, máme nejaké dáta, ktoré môžeme rozdeliť do dvoch tried a cieľom je rozhodnúť, do ktorej triedy budú patriť nové dáta, ktoré do súboru pridáme. V prípade *support vector machine* je každý dátový bod reprezentovaný ako p-dimenzionálny vektor a snahou je dosiahnuť to, aby všetky tieto body sa rozdelili pomocou (p-1)-dimenzionálnej nadroviny. V tom prípade ide o *lineárny klasifikátor*. Existuje mnoho nadrovín, ktoré by sa mohli použiť na klasifikáciu dát. Cieľom však je nájsť optimálnu nadrovinu, ktorá poskytuje najväčší stupeň seperácie alebo najväčšiu hranicu medzi dvoma triedami dát. Takáto nadrovina sa vyberie tak, že vzdialenosť najbližšieho dátového bodu od hranice je na každej strane maximálna. Ak existuje takáto nadrovina nazýva sa tiež *maximálnou hraničnou nadrovinou* a klasifikátor, ktorý ju obsahuje sa nazýva *klasifikátor s maximálnou hranicou* alebo *perceptron optimálnej stability*.

Na obrázku 11 môžeme vidieť, niekoľko nadrovín, ktoré rozdeľujú dáta. Úlohou je nájsť optimálnu nadrovinu.



Obr. 11: Nadroviny v SVM[12]

6.3 Lineárne SVM

Sú dané nejaké trénovacie dáta $\mathbb{D}$, je to vlastne súbor n bodov, pre ktoré platí:

$$\mathbb{D} = \{(x_i, y_i) \mid x_i \in \mathbb{R}^p, y_i \in \{-1, 1\}\}_{i=1}^{n}, \tag{11}$$

kde y_i je buď 1 alebo -1, podľa toho, do ktorej triedy patrí bod x_i. Každý bod x_i je p-dimenzionálny vektor reálnych čísiel. Cieľom je nájsť nadrovinu s maximálnou hranicou, ktorá rozdelí body podľa toho či sa $y_i = 1$ alebo $y_i = -1$. Každá nadrovina sa môže zapísať ako súbor bodov x, ktoré spĺňajú:

$$w \cdot x - b = 0, \tag{12}$$

kde $\cdot$ znamená skalárny súčin a w normálový vektor k nadrovine. Parameter $\frac{b}{||w||}$ určuje posun nadroviny pozdĺž normálového vektora w. Ak sú dajú trénovacie dáta lineárne rozdeliť, vyberajú sa dve nadroviny, ktoré rozdeľujú dáta tak, že nie sú medzi nimi žiadne body, a potom sa maximalizuje ich odstup. Oblasť nimi ohraničená sa nazýva *hranica*. Tieto nadroviny môžeme popísať nasledujúcim rovnicami:

$$w \cdot x - b = 1 \tag{13}$$

a

$$w \cdot x - b = -1. \tag{14}$$

Po použití geometrie dostaneme, že vzdialenosť týchto dvoch nadrovín je $\frac{2}{||w||}$ a chceme aby bolo w čo najmenšie. Ako ďalšie obmedzenie pridáme podmienku, aby nám dátové body nepadali do hranice. Pre každé i platí:

$$w \cdot x_i - b \geq 1, \tag{15}$$

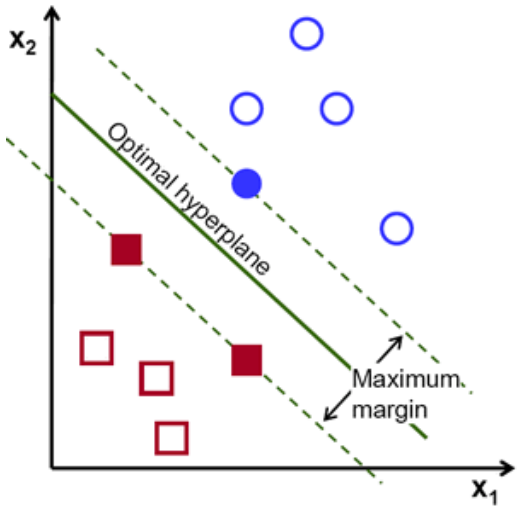
pre x_i z prvej triedy a

$$w \cdot x_i - b \leq -1, \tag{16}$$

pre x_i z druhej triedy. To môžeme prepísať ako:

$$y_i(w \cdot x_i - b) \geq 1, \tag{17}$$

pre všetky $1 \leq i \leq n$. Výsledkom všetkého je optimalizačný problém. Chceme minimalizovať $||w||$ za podmienok, že pre každé $i = 1, \ldots, n$ platí $y_i(w \cdot x_i - b) \geq 1$. Na obrázku 12 môžeme vidieť nadrovinu s maximálnou hranicou a hranice SVM. SVM je naučené pomocou dát z dvoch tried. Dátové body ležiace priamo na hranici sa nazývajú *support vectors*.



Obr. 12: Nadrovina s maximálnou hranicou[12]

6.3.1 Optimalizácia SVM

Optimalizačný problém, uvedený v predchádzajúcej časti, je ťažký na riešenie, pretože závisí na $||w||$, normálovom vektore w, ktorý zahŕňa druhú mocninu. Našťastie môžeme zmeniť rovnicu substituovaním $||w||$ výrazom $\frac{1}{2}||w||^2$, čo je podľa matematických pravidiel správne. Nazýva sa to *kvadratický programátorský optimalizačný problém*. Jeho účelom je nájsť spôsoby ako správne programátorsky optimalizovať kvadratickú funkciu. Dostaneme teda:

$$\arg\min_{(w,b)} \frac{1}{2}||w||^2 \tag{18}$$

za podmienok, že pre každé $i = 1, \ldots, n$ platí $y_i(w \cdot x_i - b) \geq 1$. Zavedieme Lagrangeove násobky α a náš problém môžeme vyjadriť ako:

$$\arg\min_{(w,b)} \max_{\alpha \geq 0} \{\frac{1}{2}||w||^2 - \sum_{i=1}^{n} \alpha_i [y_i(w \cdot x_i - b) - 1]\} \tag{19}$$

a tým pádm hľadamé stacionárne body. Na všetkých bodoch, ktoré môžu byť vyjadrené ako $y_i(w \cdot x_i - b) - 1 > 0$ nezáleží, pretože musíme nastaviť korešpondujúce α_i na nulu. Tento problém, môže byť teraz vyriešený pomocou štandardných kvadratických programovacích techník. Karush–Kuhn–Tuckerova podmienka [17] nám vyjadruje, že riešenie môžeme vyjadriť ako lineárnu kombináciu trénovacích vektorov:

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i. \tag{20}$$

Iba niekoľko α_i bude väčších ako nula. Korešpondujúce body x_i sú skutočné *support vectors*, ktoré ležia priamo na hranici a spĺňajú $y_i(w \cdot x_i - b) = 1$. Z toho môže odvodiť, že *support vectors* tiež spĺňajú:

$$y_i(w \cdot x_i - b) = 1/y_i = y_i \Leftrightarrow b = w \cdot x_i - y_i \tag{21}$$

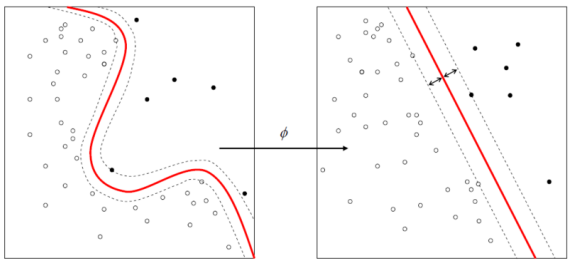
čo umožňuje definovať b. V praxi sa skôr používa vyjadrenie cez počet *support vectors* N_{SV}:

$$b = \frac{1}{N_{SV}} \sum_{i=1}^{N_{SV}} (w \cdot x_i - y_i). \tag{22}$$

6.4 Nelineárne SVM

Ako už bolo spomínané, nelineárne SVM využíva metódu *kernel trick*[16] na získanie maximálne hraničnej nadroviny. Algoritmus je takmer rovnaký, akurát každý skalárny súčin je nahradený nelineárnou kernel funkciou. Toto umožňuje algoritmu aby maximálnu hraničnu nadrovinu vtesnal do transformovaného priestoru. Transformácia býva nelineárna a transformovaný priestor je viac dimenzionálny. Medzi základné kernely patrí:

- **Polynomiálny, homogénny** = $k(x_i, x_j) = (x_i \cdot x_j)^d$
- **Polynomiálny, nehomogénny** = $k(x_i, x_j) = (x_i \cdot x_j + 1)^d$
- **Gaussova funkcia s radiálnym základom (RBF)** = $k(x_i, x_j) = \exp(-\gamma||x_i - x_j||^2)$, $\gamma > 0$ alebo $\gamma = 1/2\sigma^2$
- **Hyperbolický tangent** = $k(x_i, x_j) = \tanh(\kappa x_i \cdot x_j + c),\ k > 0,\ c < 0$



Obr. 13: Kernel trick[24]

6.5 SVM v OpenCV

V tejto práci bola použitá metóda `SVM` z knižnice OpenCV. Vlastnosti SVM môžeme nastaviť pomocou niekoľkých parametrov. Vlastnosti SVM sa nastavujú pomocou metódy `SVMParams`, ktorá prijíma nasledujúce parametre:

- **svm-type** - typ formulácie SVM, musíme zohľadňovať, či sa dáta dajú separovať lineárne alebo nie. Máme na výber z nasledujúcich možností:
  - **C-SVC** - C-Support Vector Classification. n-triedna klasifikácia pre ($n \geq 2$). Umožňuje nedokonalú separáciu tried s penalizáciou pre odľahlé hodnoty pomocou násobku parametra C.

  - **NU-SVC** - ν-Support Vector Classification. n-triedna klasifikácia s možnosťou nedokonalej separácie. Parameter ν (hodnoty 0 až 1) nastavuje hladkosť hranice rozhodovania namiesto parametra C.
  - **ONE-CLASS** - Všetky trénovacie dáta sú z rovnakej triedy, SVM vytvorí hranicu, ktorá oddelí triedu od zvyšneho priestoru.
  - **EPS-SVR** - ϵ-Support Vector Regression. Vzdialenosť medzi vektormi príznakov z trénovacej množiny a korešpondujúcej nadroviny musí byť menšia ako parameter p. Pre odľahlé body sa používa penalizácia podľa parametra C.
  - **NU-SVR** - ν-Support Vector Regression. Parameter ν je použitý namiesto parametra C.

- **kernel-type** - typ kernela SVM, vyberáme ním vlastne spôsob, akým sa budú mapovať dáta z trénovacej množiny, aby sa zlepšila schopnosť separácie.
  - **LINEAR** - lineárny kernel. Nevykonáva sa sem žiadne mapovanie a lineárne regresia sa vykonáva v pôvodnom priestore. Patrí k najrýchlejším možnostiam.
  - **POLY** - polynomiálny kernel.
  - **RBF** - funkcia s radiálnym základom. Je to najlepšia možnosť vo väčšine prípadov.
  - **SIGMOID** - kernel hyperbolického tangentu.

- **term-crit** - nastavenie kritérií pre iteračnú trénovaciu procedúru SVM, ktoré čiastkovo riešia problém obmedzeného kvadratického optimalizačného problému. Nastavuje sa pomocou nasledovných parametrov:
  - **type** - špecifikácia terminačného kritéria. Na výber je z nasledujúcich:
    - **TERMCRIT-ITER** - algoritmus sa zastaví keď dosiahne nastavenú maximálnu hodnotu iterácie.
    - **TERMCRIT-EPS** - algoritmus sa zastaví keď presnosť algoritmu klesne pod hodnotou nastavenú pomocou epsilonu.
    - **TERMCRIT-ITER+TERMCRIT-EPS** - algoritmus sa zastaví keď je splnená jedna z predchádzajúcich podmienok.
  - **max-iter** - maximálny počet iterácií algoritmu.
  - **epsilon** - požadovaná presnosť algoritmu.

7 Implementácia

Jedným z hlavných cieľov tejto práce bolo vytvoriť pomocou knižnice OpenCV aplikáciu, ktorá bude obsahovať algoritmus na detekciu objektov v obraze, a tiež algoritmus strojového učenia s učiteľom. Po preštudovaní príbuzných prací a dohode s vedúcim tejto práce bol vybraté algoritmy HoG pre detekciu objektov a algoritmy SVM pre klasifikáciu. Výsledná aplikácia je implementovaná v jazyku C/C++. Cieľom aplikácie je na zadanom vstupnom obrázku určiť obsadené a voľné parkovacie miesta.

Algoritmus vykonáva niekoľko krokov kým dospeje k cieľu. Najskôr sa na zadanom obrázku odstráni skreslenie šošovky, následne sa na obrázok aplikuje perspektívna transformácia, po ktorej je obrázok pripravený na vyrezanie jednotlivých parkovacích miest. V ďaľších krokoch dochádza k vypočítaniu histogramu orientovaných gradientov pre jednotlivé parkovacie miesta. K tomu sa používa algortimus HoG, ktorý je implementovaný v knižnici OpenCV. Konkrétne sa na to používa metóda `compute`. Dáta získané z HoGu sa priebežne ukladajú do pamäte. Keď sú dáta získané je potrebné nimi naplniť SVM klasifikátor. Opäť bola použitá implementácia z knižnice OpenCV. V tomto mieste sa algortimus delí na dve fázy. V trénovacej fázi klasifikátora sa dáta z HoG algoritmu ukladajú postupne do dvojrozmerého poľa, ktoré je uložené v globálnej pamäti. Do poľa sa postupne pridávajú dáta tak, ako sa spracúvajú samotné obrázky parkovacích miest. Nakoniec je toto pole spoločne s ďalším poľom, ktoré obsahuje značky prevedené na maticu, dátový typ `Mat`. Pole značiek obsahuje informácie o tom, či je miesto obsadené alebo voľné. Keďže ide o spôsob učenia s učiteľom, tieto značky sú dôležité, pretože práve vďaka nim určíme klasifikátoru informácie o obsadenosti. Trénovanie klasifikátora prebieha pomocou metody `train`. V testovacej fázi sa dáta z HoG algoritmu ukladajú priamo do matice, vznikne teda matica pre každé parkovacie miesto. Pomocou metódy `predict` sa jednotlivé matice posielajú do SVM klasifikátora, ktorý ako výstupný parameter vracia informáciu o tom, či je miesto obsadené alebo voľné. Na základe týchto informácií je potom vykreslený výsledok na vstupný obrázok a je zobrazený užívateľovi. Obrázok sa taktiež uloží na disk.

Aby bola aplikácia dostatočne jednoduchá, bude sa vstupný obrázok predávať pomocou parametra pri spustení. Aplikácia tiež vypisuje do konzoly kontrolné informácie o tom, v ktorom kroku algoritmu sa práve nachádza.

7.1 Organizácia kódu

Zdrojové kódy aplikácie sú rozdelené do nasledujúcich súborov:

- *RemovingLensDistortion.cpp* - implementácia algortimu pre odstránenie skreslenia šošovky,
- *PerspectiveTransform.cpp* - implementácia metódy pre perspektívnu transformáciu,
- *CutParkingSpaces.cpp* - implementácia algoritmu pre získanie parkovacích miest,
- *SVM.cpp* - obsahuje metódy pre trénovanie a testovanie klasifikátora, ktoré zahŕňajú aj výpočet histogramu orientovaných gradientov a taktiež metódu pre vykreslenie výsledkov,
- *main.cpp* - metódy pre načítavanie a ukladanie obrázkov, a tiež volanie ostatných funkcií a zobrazenie výsledkov.

8 Testovanie

Pri testovaní aplikácie je dôležité zamerať sa na to, aká bude úspešnosť pri snímkoch parkoviska, ktoré sú zachytené za rôznych svetelných a klimatických podmienok. Tiež je dôležité testovať, ktorý kernel SVM dáva najlepšie výsledky, a teda je najviac optimalizovaný pre daný problém. Z toho dôvodu bolo vykonaných niekoľko testov.

Parametre HoG deskriptora boli nastavené po dobu celého testovania nasledovne: *veľkosť okna* = 96x96, *veľkosť bloku* = 16x16, *krokovanie bloku* = 8x8, *veľkosť bunky* = 8x8, počet kanálov = 9. Zvyšné nastavenie parametrov bolo ponechané na pôvodných hodnotách.

8.1 Prvý test

V prvom teste, bol najskôr vytvorený trénovací súbor, ktorý obsahoval 216 snímkov jednotlivých parkovacích miest. Z tohto počtu bolo 108 obsadených a 108 voľných parkovacích miest. Súbor obsahoval obrázky zo všetkých radov parkoviska. Bolo to dôležité, pretože miesta v prvom rade podliehajú minimálnemu skresleniu oproti miestam v poslednom rade. Obrázky však nezachytávali parkovisko v rôznych klimatických a svetelných podmienkach, nie sú teda ovplyvnené dažďom alebo snehom. Neboli zahrnuté ani svetelné podmienky, teda žiadne skreslenie vyvolané snímkami v noci, alebo ráno pri východzajúcom slnku. Taktiež neboli zahrnuté obrázky zosnímané pri umelom osvetlení alebo za priameho slnka, ktoré spôsobuje tiene automobilov na neobsadených miestach. Každý obrázok parkovacieho miesta mal rozmery 96x96 pixelov.

Parametre SVM boli nastavené na: *typ SVM* = **C-SVC**, *typ kernelu* - **LINEAR**, parameter C = **3**, kritéria boli nastavené na *maximálny počet iterácií* **1000**.

Test prebehol na množine 20 obrázkov, ktoré zachytávali rôzne klimatické a svetelné podmienky spomínané vyššie. Celkovo sa teda testovala detekcia obsadenosti na 1120 parkovacích miestach s úspešnosťou 85,54%. Z tohto počtu bolo 534 parkovacích miest voľných a správne detekovaných bolo 379 parkovacích miest, čo predstavuje úspešnosť detekcie 70,97%. Zvyšné miesta boli obsadené, to znamená 586 obsadených parkovacích miest, pričom správne detekovaných bolo 579 parkovacích miest, t. j. úspešnosť 98,81%. Z tohto testovania vyplýva, že systém si počínal zle pri detekcii voľných miest. To však mohlo byť spôsobené tým, že sa testovali nočné snímky parkoviska, kedy tam nebol zaparkovaný žiaden automobil. Preto pre ďalšie testy bude potrebné zmeniť testovaciu množinu. Tomu je venovaný štvrtý test. V tabuľke je táto konfigurácia označená ako SVM_1.

Obr. 14: Príklad voľných parkovacích miest z trénovacej množiny

Obr. 15: Príklad obsadených parkovacích miest z trénovacej množiny

8.2 Druhý test

V druhom teste bola opäť použitá rovnaká trénovacia množina ako v prvom teste. Testoval sa ďalší typ kernelu SVM; *typ SVM* bol ponechaný na **C-SVC** spoločne aj s parametrom *C*=**3**, avšak *typ kernelu* bol zvolený **RBF** teda *funkcia s radiálnym základom*. Potrebný parameter *gamma* bol nastavený na **0.1**.

Na testovanie bola použitá rovnaká testovacia množina ako v prvom prípade. Celková úspešnosť detekcie na 1120 parkovacích miestach bola 70,27%. Pri detekcii voľných obsadených miest bola detekcia veľmi slabá, z 534 parkovacích miest sa správne detekovalo iba 201, čo je úspešnosť iba 37,64%. Na druhej strane, detekcia obsadených parkovacích miest bola 100% úspešná, bolo detekovaných všetkých 586 parkovacích miest. Celkovo je to veľmi zlý výsledok a úspešnosť detekcie obsadených parkovacích miest môže byť čisto náhodná. Preto na základne výsledkov tohto testu, môžeme úsudiť, že kernel RBF je nevyhovujúci pre tento typ úlohy. V tabuľke je táto konfigurácia označená ako SVM_2.

8.3 Tretí test

Aj v teste číslo tri, sa použila rovnaká trénovacia množina ako v predchádzajúcich dvoch prípadoch. Opäť sa testoval ďalší kernel SVM, v tomto prípade to bol *typ kernelu* = **POLY** a potrebné parametre boli nastavené nasledovne: *degree* = **5**, *gamma* = **0.1**, *coef0* = **0**. *Typ SVM* bol rovnaký ako v predchádzajúcich prípadoch a to **C-SVC** a parameter *C* = **3**.

Testovanie opäť prebehlo na rovnakej testovacej množine ako predtým, pričom teraz bola celková úspešnosť na 1120 parkovacích miestach 84,91%. Z 534 voľných parkovacích miest bolo správne detekovaných 366, čo predstavuje úspešnosť 68,54%. Z 586 obsadených parkovacích miest bolo správne detekovaných 585, čo predstavuje úspešnosť 99,83%.

Z výsledkov môžeme úsudiť, že riešeniu nášho problému najlepšie vyhovujú lineárny a polynomiálny kernel. Avšak algoritmus stále dáva pomerne zlé výsledky, preto treba v ďalších testoch zmeniť trénovaciu množinu, rozšíriť o ďalšie prípady, kedy je parkovisko zachytené v rôznych klimatických a svetelných podmienkach. K tomu pristúpime v ďalších testoch. V tabuľke je táto konfigurácia označená ako SVM_3.

8.4 Štvrtý test

Najskôr bol vytvorená nová trénovacia množina príkladov. K pôvodným 216 snímkom parkovacích miest boli pridané ďalšie snímky zachytávajúce v parkovisko v rôznych klimatických a svetelných podmienkach. Nová trénovacia množina obsahovala 352 snímkov parkovacích miest, z toho bolo 176 snímkov voľných parkovacích miest a 176 snímkov obsadených parkovacích miest. Nové snímky boli zo všetkých radov parkoviska a taktiež mali rozmery 96x96 pixelov ako pri prvej trénovacej množine.

Nastavenie parametrov SVM bolo zvolené takto: *typ SVM* = **C-SVC**, *typ kernelu* = **LINEAR**, *C* = **3**. Úspešnosť detekcie na množine parkovacích miest bola 95,63%, bolo teda detekovaných 1071 z 1120 parkovacích miest. Úspešnosť detekcie na množine voľných parkovacích miest bola 97,94%, detekovaných bolo 523 z 534 voľných parkovacích miest. Pri detekcie obsadených parkovacích miest bolo detekovaných 548 z 586 obsadených parkovacích miest, čo predstavuje úspešnosť 93,52%.

Tento test potvrdil, že pre dosiahnutie výsledkov, ktoré sa blížia k 100% treba zvoliť v prvom rade správne zloženie trénovacej množiny, ktorá by mala zachytávať všetky možné varianty počasia, v ktorom sa parkovisko môže vyskytovať. Dôležitý je aj čas a do trénovacej množiny je potrebné zahŕnuť aj snímky parkoviska v rôznych intervaloch dňa a noci. V tabuľke je táto konfigurácia označená ako SVM_4.

8.5 Piaty test

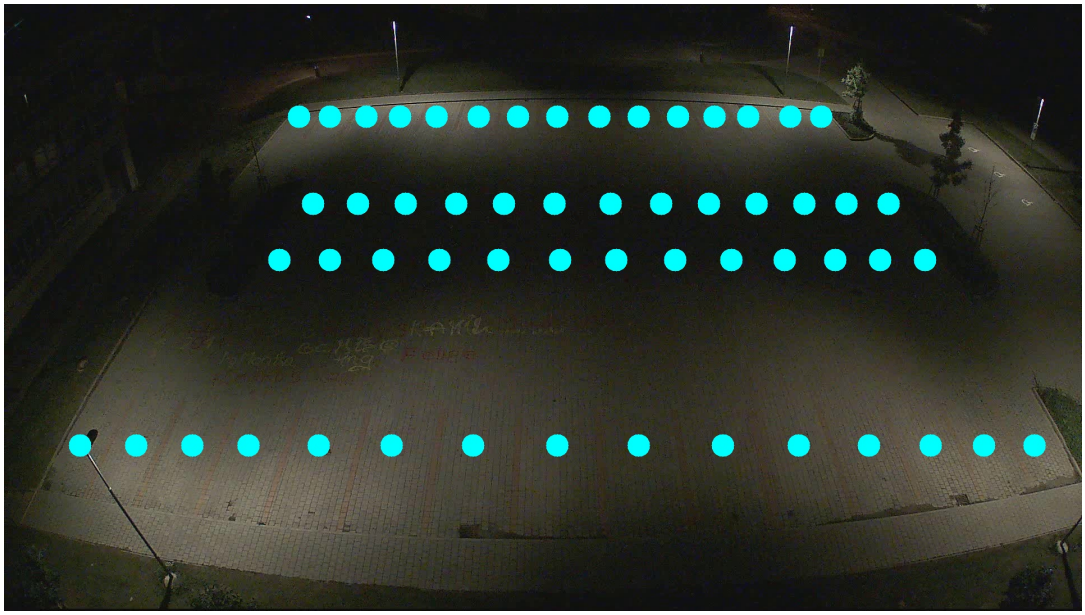
V piatom teste bola použitá trénovacia množina príkladov, ktorá obsahovala 352 snímkov. Parametre boli zvolené opäť *typ SVM* = **C-SVC**, *typ kernelu* = **POLY**, parameter *C* = **3**, parameter *degree* = **5**, parameter *gamma* = **0.1**, parameter *coef0* = **0**.

Úspešnosť detekcie na množine parkovacích miest bola 98,75%. Bolo detekovaných 1106 z 1120 parkovacích miest. Voľných parkovacích miest bolo detekovaných 528 z 534, úspešnosť detekcie teda bola 98,88%. Obsadených parkovacích miest bolo detekovaných 578 z 586 parkovacích miest, úspešnosť teda bola 98,63%. Z tohto testu vyplynulo, že

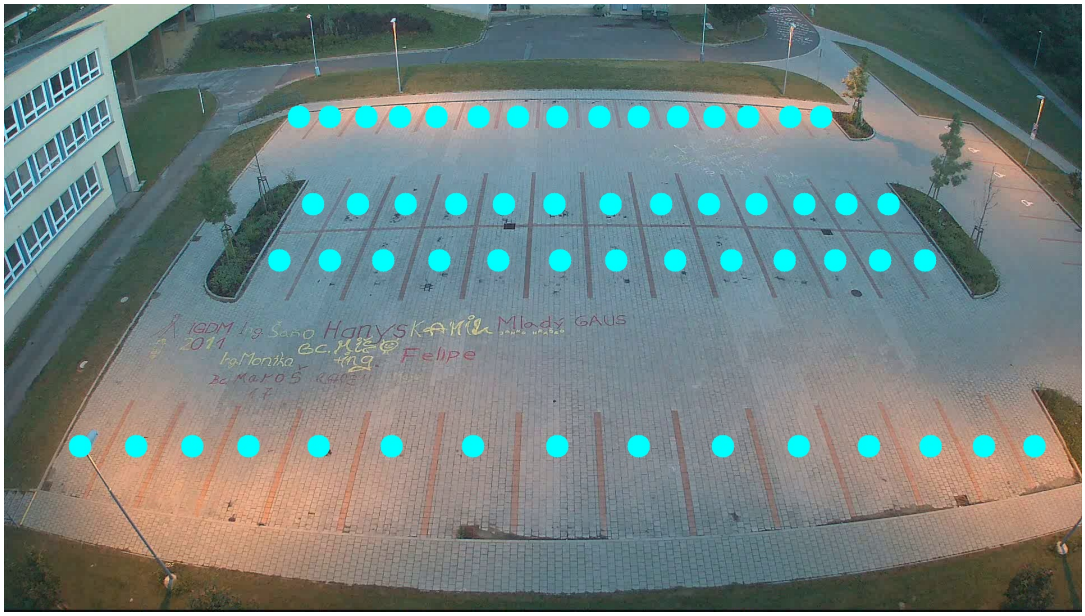
polynomiálny kernel ponúka ešte lepšie výsledky ako lineárny kernel a dáva veľmi dobré výsledky. V tabuľke je táto konfigurácia označená ako SVM_5.

Typ	Obsadené	Detek.	Per.	Prázdne	Detek.	Per.	**Úspešnosť**
SVM_1	586	579	98,81%	534	379	70,97%	**85,54%**
SVM_2	586	586	100%	534	201	37,64%	**70,27%**
SVM_3	586	585	99,83%	534	366	68,54%	**84,91%**
SVM_4	586	548	93,52%	534	523	97,94%	**95,63%**
SVM_5	586	578	98,63%	534	528	98,88%	**98,75%**

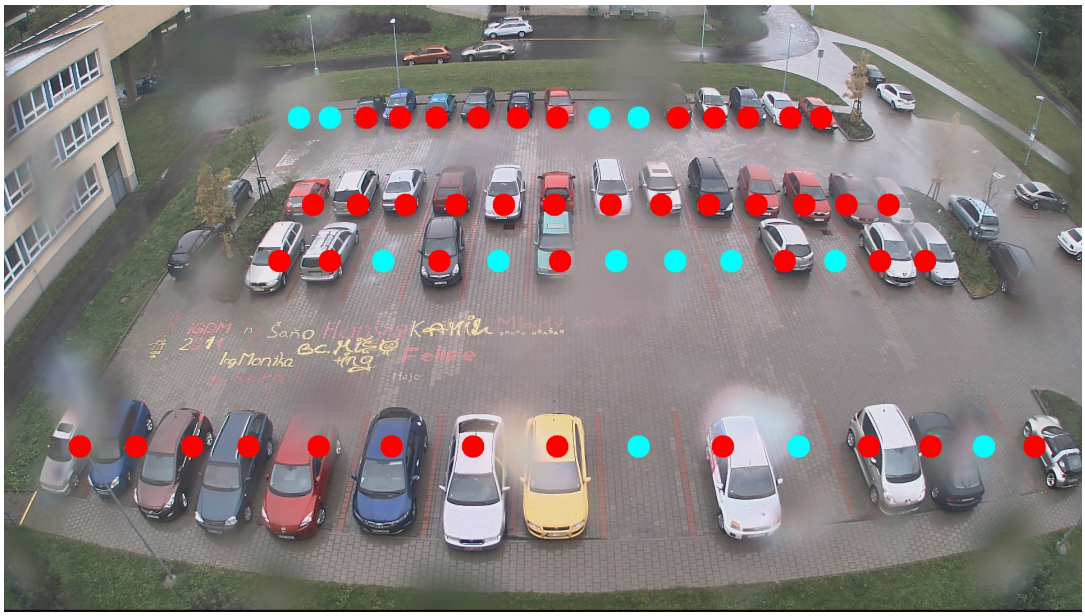
Tabuľka 1: Porovnanie jednotlivých výsledkov



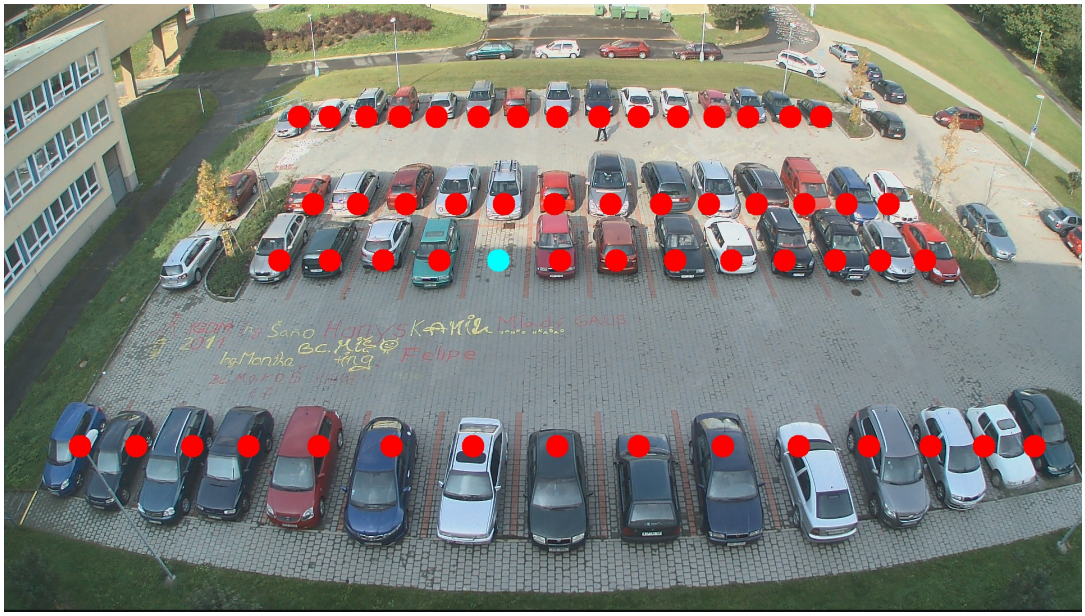
Obr. 16: Výsledky detekcie na nočnom snímku



Obr. 17: Výsledky detekcie na neobsadenom parkovisku



Obr. 18: Výsledky detekcie na snímku v daždi



Obr. 19: Výsledky detekcie na obsadenom parkovisku

Obr. 20: Výsledky detekcie na zasneženom snímku

9 Záver

Ciele bakalárskej práce uvedené v zadaní sa podarilo splniť. Práca teoreticky oboznamuje s technikou detekcie objektov v obraze, a tiež teoreticky vysvetľuje princíp strojového učenia s učiteľom. V práci je vysvetlené ako funguje algoritmus *histogramu orientovaných gradientov*, ktorý sa použil na detekciu objektov v obraze. Rovnako je aj vysvetlený algoritmus *support vector machine*, ktorý bol vybratý na klasifikovanie z pomedzi množstva algoritmov strojového učenia. Ďalším cieľom bolo implementovať aplikáciu na detekovanie obsadenosti parkovacích miest, ktorá bude využívať tieto dva algortimy. Aplikácia bola implementovaná v jazyku C++ a využíva funkcie knižnice OpenCV. Následne bola aplikácia otestovaná a v práci sú vyhodnotené výsledky.

Pri testoch sa preukázalo, že úspešný výsledok záleží od zloženia trénovacej množiny, do ktorej treba zahrnúť snímky parkoviska v rôznych svetelných a klimatických podmienkach. Testovali sa dve trénovacie množiny: prvá, ktorá obsahovala snímky parkoviska len za slnečného počasia a v denných intervaloch dňa. Druhá trénovacia množina, obsahovala snímky parkoviska v rôznych klimatických podmienkach, a taktiež v denných aj nočných intervaloch dňa. Pri testoch sa potvrdilo, že zloženie druhej testovacej množiny je vyhovujúce, pretože detekcia bola úspešná na takmer 99%. Môžeme preto povedať, že *support vector machine* a správna množina príkladov je vyhovujúca pre riešenie zadanej úlohy.

Riešenie však ponúka množstvo miest, v ktorých môže byť vylepšené alebo rozšírené. Jednou z možností by bolo použiť inú metódu na detekciu objektov v obraze, alebo dokonca použiť iný algoritmus strojového učenia. Tiež by bolo zaujímavé aplikáciu detailenejšie otestovať napríklad v zimnom období, v ktorom môže sneh spôsobiť veľké skreslenie. Bolo by aj vhodné toto riešenie vyskúšať na inom parkovisku, ale preto by sa muselo vykonať nové mapovanie.

Michal Šamaj

10 Literatúra

[1] Duda, Hart, Stork: *Pattern Classificiation*, 2000. ISBN: 978-0471056690.

[2] Mohri, Rostamizadeh, Talwalkar: *Foundations of Machine Learning*, 2012. ISBN: 978-0262018256.

[3] Constantine Papageorgiou a Tomaso Poggio, *A trainable system for object detection*. *Int. J. Comput. Vision*, 38(1):15 - 33, jún 2000.

[4] H. Schneiderman a T. Kanade, *A statistical method for 3d object detection applied to faces and cars.* V *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vydanie 1, strany 746 - 751 vyd.1, 2000.

[5] Bernd Heisele, Ivaylo Riskov, a Christian Morgenstern, *Components for object detection and identification.* V *Toward Category-Level Object Recognition*, strany 225-237, 2006.

[6] S. Hinz, *Detection and counting of cars in aerial images*. V *Image Processing, ICIP 2003. Proceedings. 2003 International Conference*, 3. vydanie, strany 997-1000. 2003.

[7] Tao Zhao Ram, Tao Zhao, a Ram Nevatia, *Car detection in low resolution aerial images*. V *Image and Vision Computing*, strany 710-717. 2001.

[8] Stanley M. Bileschi, Brian Leung, a Ryan M. Rifkin, *Component-based cardetection*. V *in Street Scene Images*. MIT. 2004.

[9] Paul van Walree, *Distortion*, V *Photographic optics*. 2009.

[10] Bilinear Interpolation Definition, [online] dostupné z: <http://www.pcmag.com/encyclopedia/term/38607/bilinear-interpolation> [apríl 2014]

[11] Brown, Duane C., *Decentering distortion of lenses*. V *Photogrammetric Engineering*. 32 (3): 444–462.

[12] OpenCV [online] dostupné z: <http://opencv.org/> [apríl 2014]

[13] Navneet Dalal a Bill Triggs, *Histograms of Oriented Gradients for Human Detection*. INRIA Rhone-Alps, 655 avenue de l'Europe, Montbonnot 38334, France

[14] Cortes, C. a Vapnik, V., *Support-vector networks. Machine Learning* 20 (3): 273, 1995.

[15] William H.; Teukolsky, Saul A.; Vetterling, William T.; Flannery, B. P., *Section 16.5. Support Vector Machines*. V *Numerical Recipes: The Art of Scientific Computing*, 3. vydanie, New York: Cambridge University Press, 2007. ISBN 978-0-521-88068-8.

[16] Aizerman, Mark A.; Braverman, Emmanuel M.; a Rozonoer, Lev I., *Theoretical foundations of the potential function method in pattern recognition learning, Automation and Remote Control* 25: 821–837. 1964

[17] Kuhn, H. W. a Tucker, A. W., *Nonlinear programming, Proceedings of 2nd Berkeley Symposium*. Berkeley: University of California Press, strany 481–492. 1951

[18] Obrázok dostupný z: <http://mrl.cs.vsb.cz/publications/fusek_adaboost_parking_2013.pdf> [apríl 2014]

[19] Obrázok dostupný z: <http://www.popularmechanics.com/technology/gadgets/news/smart-parking-systems-steer-drivers-to-open-spaces> [apríl 2014]

[20] Obrázok dostupný z: <http://en.wikipedia.org/wiki/Distortion_(optics)> [apríl 2014]

[21] Obrázok dostupný z: <http://en.wikipedia.org/wiki/Bilinear_interpolation> [apríl 2014]

[22] Obrázok dostupný z: <http://ryanlei.wordpress.com/2011/03/09/> [apríl 2014]

[23] Obrázok dostupný z: <http://www.cnblogs.com/slysky/archive/2012/04/25/2470057.html> [apríl 2014]

[24] Obrázok dostupný z: <http://en.wikipedia.org/wiki/Support_vector_machine> [apríl 2014]

A Obsah priloženého CD

Obsah priloženého CD:

- Testovacia aplikácia `Detection.exe`
- Zložka so zdrojovými kódmi `/sourceCode/`
- Zložka s testovacími obrázkami `/testImages/`
- Návod na spustenie aplikácie `ReadMe.txt`
- Textová časť bakalárskej práce vo formáte PDF `BP_SAM0036.pdf`